Delft University of Technology

DSE Spring 2023

AE3200

Verification of Simulation

Author:
Matei Dinescu (5260248)
Jasper Geijsberts (5331587)
Ian Maes (5216931)
Serban Nedelcu (5324491)
Andreas Van Parys (5089263)
Lennart van der Peet (5280168)
Nikolaus Oliver Ricker Chong (5274737)
Kyle Scherpenzeel (5310210)
Carl Spichal (5222397)
Mathijs Vereycken (5324661)

June 23, 2023

Contents

Chapter 1

Introduction

The following document contains the unit, system and module tests of all the codes performed during the final phase of the DSE. All the tests will follow a specific template when reporting them. They will be in a table with five columns similar to Table 1.1, which is an example. The first column contains an ID composed of 2 letters which are: MO (Moon Orbits), EO (Earth Orbits), CM (Coverage Model), DS (Dynamic Simulation), LB (Link Budget), ST (Sensitivity analysis Trade-off), UE (User Error) followed by a number: the number of the function in order of appearance in the code and finally a letter which is given if there are multiple unit tests made on the same function. Next to the ID, the name of the function is given. The other columns are self-explanatory and can be seen in Table 1.1.

Table 1.1: Unit Tests Template Table

Function	Function Description	Unit Test Description	Technique Used	Results
MO-1-A `transf()`	Coordinates transformation from frame A to B	Calculation of the transformation at a specific time	By hand	Same result at that time
MO-1-B `transf()`	Coordinates transformation from frame A to B	Checking size of the final matrix	Unit test function in code	Size it should be

Chapter 2

Streets of Coverage

Table 2.2: System tests for the streets of coverage estimation model.

Function	Function Description	Unit Test Description	Technique Used	Results
MO-3-A `loop()`	Loop through multiple altitudes and street satellite counts with an array output of all combinations	Size check of final array	Unit test function in code	Size it should be
MO-3-B `loop()`	Loop through multiple altitudes and street satellite counts with an array output of all combinations	Hand calculation of one option of the array	Unit test function in code	Same result as array
MO-3-C `loop()`	Loop through multiple altitudes and street satellite counts with an array output of all combinations	Check altitude of 0 km to check for 'Fail' output	Unit test function in code	'Fail' output received
MO-3-D `loop()`	Loop through multiple altitudes and street satellite counts with an array output of all combinations	Check altitude of 1×10^8 km to check for 2 orbit planes output.	Unit test function in code	Two orbital planes as output

Table 2.1: Unit tests for the streets of coverage estimation model.

Function	Function Description	Unit Test Description	Technique Used	Results
MO-1-A `incl()`	Minimum inclination and coverage angle calculation	Hand calculation of the inclination angle at a specific altitude	Unit test function in code	Same result at that altitude
MO-1-B `incl()`	Minimum inclination and coverage angle calculation	Hand calculation of the coverage angle at a specific altitude	Unit test function in code	Same result at that altitude
MO-2-A `n_sats()`	Number of satellites and orbit planes calculation	Hand calculation of the number of orbit planes at a specific altitude and street satellite count	Unit test function in code	Same result at that altitude
MO-2-B `n_sats()`	Total number of satellites and orbit planes calculation	Hand calculation of the number of satellites at a specific altitude and street satellite count	Unit test function in code	Same result at that altitude

Chapter 3

Earth Orbit

Table 3.1: Unit Tests for the Earth Orbits Model

Function	Function Description	Unit Test Description	Technique Used	Results
EO-1-A `moon_sat_cart_pos()`	Find coordinates in time of the Moon and the six Satellites	Third column (z-position) of all the satellites and Moon position should be zero	Unit test function in code, print if wrong for 20 time steps	Nothing is printed
EO-1-B `moon_sat_cart_pos()`	Find coordinates in time of the Moon and the six Satellites	LL1, Moon and LL2 should be on a straight line through the origin at all times	Unit test function in code, print if wrong for 20 time steps	Nothing is printed until 10^{-13}
EO-2-A `transf()`	Find coordinates of S3-S6 in R	Any vector (x, y, 0) in R_r should be ($x\cos(i_r)$, y, $x\sin(i_r)$) after the transformation	Unit test function in code, print if wrong for 20 time steps	Nothing is printed
EO-3-A `Dist_req()`	Find distance between satellites and Moon at any time	A satellite should not enter the Moon's SOI	Print statement if it does enter it	Nothing is printed

Table 3.2: Module Tests for the Earth Orbits Model

Function	Module Test Description	Technique Used	Results
EO-4 Module Test	Extreme Value : $\mu_E = 0$	Check coordinates at 20 time steps	All satellites and Moon do not move
EO-5 Module Test	Extreme Value : $R_{SOI} = 66.1 \times 10^{12}$ m	Check if the length of the array of the times when the four satellites are in the SOI is equal to 80000 (20000 time steps)	Length is 80000
EO-6 Module Test	Extreme Value : $\alpha_0 = 0$ deg	Check if the coordinates of S3 to S6 are the same to the Moon coordinates when the orbits intersect (P/4 and 3P/4)	They are indeed the same

Table 3.3: System Tests for the Earth Orbits Model

Function	System Test Description	Technique Used	Results
EO-7 System Test	Calculating the three spherical coordinates of S3 at 5 different time periods and comparing it to the values they should be.	Function in code	All angles are what they should be
EO-8 System test	Plotting the Moon and six satellites at 400 time steps to check if everything is correct	Function in code	Orbits are correct

Chapter 4

Link Budget

Table 4.1: Link Budget Verification

Function	Function Description	Unit Test Description	Technique Used	Results
LB-1-A `Gpeak_helical()`	Calculates gain of the transmitter	Check value of the gain at the end	Unit test function in code	Same value
LB-2-A `alpha1_over_2_helical()`	Calculates half-power angle of the helical antenna	Check value of half-power angle at the end	Unit test function in code	Same value
LB-2-B `alpha1_over_2_helical()`	Calculates half-power angle of the helical antenna	Check ZeroDivisionError of half-power angle calculation	Unit test function in code	ZeroDivision Error occurs.
LB-3-A `Lpr()`	Calculates pointing loss	Check value of pointing loss at the end	Unit test function in code	Same value
LB-3-B `Lpr()`	Calculates pointing	Check ZeroDivisionError of pointing loss calculation	Unit test function in code	ZeroDivision Error occurs.
LB-4-A `dB()`	Transforms value inserted into decibels	Check value at the end	Unit test function in code	Same value
LB-5-A `snr()`	Calculates the total SNR provided by the link	Check value of SNR at the end	Unit test function in code	Same value
LB-6-A `snr_margin()`	Calculates SNR_{margin} of the link budget	Check value of SNR_{margin} at the end	Unit test function in code	Same value
LB-7-A `S_lagrange()`	Calculates distance from side of celestial body to the lagrange point	Check value of distance at the end	Unit test function in code	Same value
LB-8-A `S()`	Calculates distance from side of earth to the desired point	Check value of distance at the end	Unit test function in code	Same value

Chapter 5

Dynamic Simulation

Table 5.1: Unit Test Dynamic Simulation

Function	Function Description	Unit Test Description	Technique Used	Results
DS-1-A orbit_param()	Sets orbital parameters to class	Checks if the parameter updates correctly	Python Unittest Module	Passed
DS-2-A final_time()	Sets final time for simulation to class	Checks if the parameter updates correctly	Python Unittest Module	Passed
DS-3-A resolution()	Sets simulation resolution to class	Checks if the parameter updates correctly	Python Unittest Module	Passed
DS-4-A mass_sat()	Sets satellite masses to class	Checks if the parameter updates correctly	Python Unittest Module	Passed
DS-5-A area_sat()	Sets satellite area to class	Checks if the parameter updates correctly	Python Unittest Module	Passed
DS-6-A c_radiation()	Sets radiation coefficient to class	Checks if the parameter updates correctly	Python Unittest Module	Passed
DS-7-A inv_orbit_param()	Test initialization with less than 2 satellites	Checks if valueerror exception is is raised	Python Unittest Module	Passed
DS-8-A inv_final_time()	Test initialization with negative final time	Checks if valueerror exception is is raised	Python Unittest Module	Passed
DS-9-A inv_resolution()	Test initialization with negative resolution	Checks if valueerror exception is is raised	Python Unittest Module	Passed
DS-10-A inv_mass()	Test initialization with negative mass	Checks if valueerror exception is is raised	Python Unittest Module	Passed
DS-11-A area_sat()	Test initialization with negative area	Checks if valueerror exception is is raised	Python Unittest Module	Passed
DS-12-A c_radiation()	Test initialization with negative radiation coeff	Checks if valueerror exception is is raised	Python Unittest Module	Passed
DS-13-A satellite_count()	Test if the amount of satellites created is correct	Checks if bodies to propagate equals length of orbital param	Python Unittest Module	Passed

DS-14-A central_bodies ()	Test of the amount of central bodies are created	Checks if the amount of central bodies equ-al the orbit parameters	Python Unittest Module	Passed
DS-15-A acceleration_ model()	Test if each satellite has 3 perturbing bodies	Check the length of each item in the acceleration model	Python Unittest Module	Passed
DS-16-A Hohmann_de-lta_V()	Test if the correct ΔV is computed for Hohmann transfer	Compares ΔV of hand calculation and code	Hand Calculation with Unittest Module	Passed
DS-17-A Inclination_de-lta_V()	Test if the correct ΔV is computed for inclinationchange	Compares ΔV of hand calculation and function	Hand Calculation with Unittest Module	Passed

Table 5.2: Module Tests for Dynamic Simulation

Function	Function Description	Unit Test Description	Technique Used	Results
DS-18-A min_max Kepler()	Tests the type of the average Kepler element range output	Tests if the output is a numpy array	Python Unittest Module	Passed
DS-18-B min_max Kepler()	Tests the type of the max Kepler element range output	Tests if the output is a numpy array	Python Unittest Module	Passed
DS-18-C min_max Kepler()	Tests the type of the SD Kepler element range output	Tests if the output is a numpy array	Python Unittest Module	Passed
DS-18-D min_max Kepler()	Tests the size of the average Kepler element range output	Test if the size is 6 for so that each elem-ent is included	Python Unittest Module	Passed
DS-18-E min_max Kepler()	Tests the size of the max Kepler element range output	Test if the size is 6 for so that each elem-ent is included	Python Unittest Module	Passed
DS-18-F min_max Kepler()	Tests the size of the SD Kepler element range output	Test if the size is 6 for so that each elem-ent is included	Python Unittest Module	Passed
DS-19-A delta_v_or-bit_maint()	Tests if the ΔV calculation is correct at the end of the simulation	Compares ΔV calculation with hand calculation	Hand calculation and Python Unit-test Module	Passed

Table 5.3: Extreme Value Tests Dynamic Simulation

Function	Function Description	Results
DS-20-A extreme_ resolution()	Look at outcome for resolution of 10000000	Semi-major axis crashed to 0 and simulation did not perform correctly
DS-21-A small_mass ()	Look at outcome for a mass of nearly 0	Semi-major axis crashed to 0 and simulation did not perform correctly
DS-21-B large_mass ()	Look at outcome for a mass of $1 \cdot 10^{18}$	Simulation still performed as intended with smaller perturbations
DS-22-A large_final _time()	Look at simulation for final time of $1 \cdot 10^{20}$	Simulation has a runtime error as time is too large
DS-22-B small_final _time()	Look at simulation for final time of 1 second	Simulation performs one resolution if bigger, or one multiple if resolution is smaller
DS-23-A large_a()	Look at simulation for semi major axis of $20 \cdot 10^{20}$	Simulation runs but perturbations are too small too see
DS-23-B small_a()	Look at simulation for semi major axis of $2 \cdot 10^{6}$	Semi-major axis quickly dropped to 0 and simulation stopped performing
DS-24-A extreme_i ()	Look at simulation for inclination above or below 180	Simulation reset initial conditions between 0-180 and performed normally
DS-25-A large_e()	Look at simulation for very large values of eccentricity	Simulation does not work and breaks
DS-25-B small_a()	Look at simulation for negative values of eccentricity	Simulation does not work and breaks
DS-26-B extreme_ AOP()	Look at simulation for values above 360 and negative values for AOP	Simulation runs nominally and resets AOP between 0-360

To be able to analyse the sensitivity of the simulation, three situations were considered. These three situations consist out of change in mass, semi-major axis and resolution. These three situations were run with the same 24 satellites as in the chosen model and their range of Kepler elements compared. This range was compared by dividing the larger sensitivity model by the smaller one. As can be seen, a larger mass and resolution improves stability of the elements, which could be desired in the later design.

Table 5.4: Kepler Element Sensitivity Large/Small

	a	e	i	ω	Ω	V
100x mass	0.99985243	0.99995343	0.99989535	0.99995673	0.99999986	1.00001048
2x a	18.46357097	4.5936315	3.75326573	0.98274801	1.08328306	1.0304849
900x resolution	0.99275369	0.99093828	0.9899416	0.99979347	0.99941231	1.00020475

Chapter 6

Coverage Model

Table 6.1: Unit tests for Coverage Model

Function	**Function Description**	**Unit Test Description**	**Technique Used**	**Results**
CM-01-A `r_size()`	Vector from origin to satellite	Checks size of vector	Unit test function in code	Passed
CM-02-A `range()`	Maximum distance to Moon surface	Checks that the range is the same as calculated by hand	Unit test function in code	Passed
CM-03-A `range_limit()`	Scenario where satellite covers more than half the Moon	Check standard value is obtained	Unit test function in code	Passed
CM-04-A range_elevation()	Elevation affects the range	Tests range is reduced by elevation	Unit test function in code	Passed
CM-05-A `isInView()`	Function to check if two points are within range	Confirms point is in view	Unit test function in code	Passed
CM-05-B `isInView()`	Function to check if two points are within range	Confirms point is not in view	Unit test function in code	Passed
CM-06-A `test_a_sat()`	Semimajor Axis of satellite	Checks error raises when small a	Unit test function in code	Passed
CM-06-B `test_a_sat()`	Semimajor Axis of satellite	Checks error raises when orbit going through surface	Unit test function in code	Passed
CM-07-A `test_e_sat()`	Eccentricity of satellite	Check error raises with negative e	Unit test function in code	Passed
CM-08-A `test_i_sat()`	Inclination of satellite	Check error raises with negative i	Unit test function in code	Passed
CM-08-B `test_i_sat()`	Inclination of satellite	Check error raises with +180 angle	Unit test function in code	Passed
CM-09-A `test_w_sat()`	Arg. Pericenter of satellite	Check error raises with negative w	Unit test function in code	Passed

CM-09-B `test_w_sat()`	Arg. Pericenter of satellite	Check error raises with +360 angle	Unit test function in code	Passed
CM-10-A `test_omega_sat()`	Ascending node of satellite	Check error raises with negative omega	Unit test function in code	Passed
CM-10-B `test_omega_sat()`	Ascending node of satellite	Check error raises with +360 angle	Unit test function in code	Passed
CM-11-A `test_nu`	True anomaly of satellite	Check anomaly resets after 360°	Unit test function in code	Passed
CM-12-A `test_elev_sat()`	Elevation of Moon	Check error raises with negative elevation	Unit test function in code	Passed
CM-13-A `r_conversion_kepler()`	Conversion from Kepler to Cartesian	Checks correct transformation	Unit test function in code	Passed
CM-13-B `r_conversion_kepler()`	Conversion from Kepler to Cartesian	Checks correct transformation	Unit test function in code	Passed
CM-13-C `r_conversion_kepler()`	Conversion from Kepler to Cartesian	Checks correct transformation	Unit test function in code	Passed
CM-13-D `r_conversion_kepler()`	Conversion from Kepler to Cartesian	Checks correct transformation	Unit test function in code	Passed
CM-14-A `test_phi()`	Longitude	Error when smaller than -180°	Unit test function in code	Passed
CM-14-B `test_phi()`	Longitude	Error when larger than 180°	Unit test function in code	Passed
CM-15-A `test_theta()`	Latitude	Error when smaller than -90°	Unit test function in code	Passed
CM-15-B `test_theta()`	Latitude	Error when larger than 90°	Unit test function in code	Passed
CM-16-A `test_h()`	Height from surface	Error when height is negative	Unit test function in code	Passed
CM-17-A `r_conversion_polar()`	Conversion from Polar to Cartesian	Checks that transformation is done correctly	Unit test function in code	Passed
CM-17-B `r_conversion_polar()`	Conversion from Polar to Cartesian	Checks that transformation is done correctly	Unit test function in code	Passed
CM-17-C `r_conversion_polar()`	Conversion from Polar to Cartesian	Checks that transformation is done correctly	Unit test function in code	Passed
CM-17-D `r_conversion_polar()`	Conversion from Polar to Cartesian	Checks that transformation is done correctly	Unit test function in code	Passed

CM-18-A `valid_input_lagrange()`	Type of Lagrange point	Checks that valid string is given	Unit test function in code	Passed
CM-19-A `test_r()`	Measures distance from origin to L1	Tests correct distance from origin for L1	Unit test function in code	Passed
CM-19-B `test_r()`	Measures distance from origin to L2	Tests correct distance from origin for L2	Unit test function in code	Passed
CM-20-A `valid_input_fix()`	Set of Cartesian elements	Checks it is a list	Unit test function in code	Passed
CM-21-A `valid_size()`	List of Cartesian elements	Checks size is 3	Unit test function in code	Passed
CM-22-A `test_a_orbit()`	Semimajor Axis of Plane	Error if small a	Unit test function in code	Passed
CM-22-B `test_a_orbit()`	Semimajor Axis of Plane	Error is crosses surface	Unit test function in code	Passed
CM-23-A `test_e_orbit()`	Eccentricity of Plane	Error if negative e	Unit test function in code	Passed
CM-24-A `test_i_orbit()`	Inclination of Plane	Error if negative i	Unit test function in code	Passed
CM-24-B `test_i_orbit()`	Inclination of Plane	Error if large i	Unit test function in code	Passed
CM-25-A `test_w_orbit()`	Arg. Pericenter of Plane	Error if negative w	Unit test function in code	Passed
CM-25-B `test_w_orbit()`	Arg. Pericenter of Plane	Error if w +360°	Unit test function in code	Passed
CM-26-A `test_omega_orbit()`	Ascending Node of Plane		Unit test function in code	Passed
CM-26-B `test_omega_orbit()`	Ascending Node of Plane		Unit test function in code	Passed
CM-27-A `test_ elev_orbit()`	Elevation from surface		Unit test function in code	Passed
CM-28-A `nsat()`	Satellites per plane	Checks satellites are positive	Unit test function in code	Passed
CM-29-A `creation_sat()`	Creation of plane with satellites	Count satellites	Unit test function in code	Passed
CM-29-B `creation_sat()`	Creation of plane with satellites	Count satellites	Unit test function in code	Passed
CM-30-A `indiv_sat()`	Creation of a single satellite	Checks the semi-major-axis of the satellite	Unit test function in code	Passed

CM-30-B `indiv_sat()`	Creation of a single satellite	Checks the eccentricity of the satellite	Unit test function in code	Passed
CM-30-C `indiv_sat()`	Creation of a single satellite	Checks the inclination of the satellite	Unit test function in code	Passed
CM-30-D `indiv_sat()`	Creation of a single satellite	Checks the argument of pericenter of the satellite	Unit test function in code	Passed
CM-30-E `indiv_sat()`	Creation of a single satellite	Checks the longitude of the ascending node of the satellite	Unit test function in code	Passed
CM-30-F `indiv_sat()`	Creation of a single satellite	Checks the elevation of the satellite	Unit test function in code	Passed
CM-31-A `sat_distr()`	Distributes all the satellites	Checks that satellite one is placed at a true anomaly of zero	Unit test function in code	Passed
CM-31-B `sat_distr()`	Distributes all the satellites	Checks that satellite one is placed at a true anomaly of 72 degrees	Unit test function in code	Passed
CM-31-C `sat_distr()`	Distributes all the satellites	Checks that satellite one is placed at a true anomaly of 144 degrees	Unit test function in code	Passed
CM-31-D `sat_distr()`	Distributes all the satellites	Checks that satellite one is placed at a true anomaly of 216 degrees	Unit test function in code	Passed
CM-31-E `sat_distr()`	Distributes all the satellites	Checks that satellite one is placed at a true anomaly of 288 degrees	Unit test function in code	Passed
CM-32-A `rel_dist_sat()`	Distributes the satellites relative to each other.	Checks that the distance between two satellites is equal to around 2×10^6 m	Unit test function in code	Passed
CM-33-A `resolution()`	Creates the resolution	Checks that the resolution is positive.	Unit test function in code	
CM-34-A `addExisting OrbitPlane()`	Adds an existing orbit plane	Checks that an existing orbit plane is added correctly by checking if the number of orbit planes is equal to one	Unit test function in code	Passed

CM-34-B `addExisting OrbitPlane()`	Add an existing orbit plane to model	Checks that plane object is added to model	Unit test function in code	Passed
CM-34-C `addExisting OrbitPlane()`	Add an existing orbit plane to model	Checks number of satellites is one	Unit test function in code	Passed
CM-35-A `add OrbitPlane()`	Creates an orbit plane in the model	Checks one plane is added to model	Unit test function in code	Passed
CM-35-B `add OrbitPlane()`	Creates an orbit plane in the model	Checks one satellite is added to model	Unit test function in code	Passed
CM-36-A `addTower()`	Creates a Moon tower	Checks no orbit plane is created	Unit test function in code	Passed
CM-36-B `addTower()`	Creates a Moon tower	Checks one tower is added to model	Unit test function in code	Passed
CM-37-A `addSatellite()`	Creates an orbiting satellite	Checks no orbit plane is created	Unit test function in code	Passed
CM-37-B `addSatellite()`	Creates an orbiting satellite	Checks one satellite is added to model	Unit test function in code	Passed
CM-38-A `addLagrange()`	Creates a satellite at a Lagrange point	Checks one satellite is added	Unit test function in code	Passed
CM-38-B `addLagrange()`	Creates a satellite at a Lagrange point	Checks two satellites are added	Unit test function in code	Passed
CM-38-C `addLagrange()`	Creates a satellite at a Lagrange point	Checks no orbit plane is created	Unit test function in code	Passed
CM-39-A `addFixPoint()`	Creates a satellite in a fix point in space	Checks no orbit plane is created	Unit test function in code	Passed
CM-39-B `addFixPoint()`	Creates a satellite in a fix point in space	Counts one element is added	Unit test function in code	Passed
CM-40-A `addExisting Module()`	Adds an existing element to the model	Counts one element is added	Unit test function in code	Passed
CM-40-B `addExisting Module()`	Adds an existing element to the model	Checks type of module is added correctly	Unit test function in code	Passed
CM-41-A `add Symmetrical Planes()`	Creates symmetrical orbital planes in model	Tests number of orbital planes	Unit test function in code	Passed
CM-41-B `add Symmetrical Planes()`	Creates symmetrical orbital planes in model	Tests number of satellites created	Unit test function in code	Passed
CM-42-A `moonCreation()`	Creates a discretised Moon	Checks that moon has enough points	Unit test function in code	Passed

CM-43-A `Coverage()`	Function to analyse coverage	Checks that coverage is only ran if there are modules	Unit test function in code	Passed
CM-44-A `plotCoverage()`	Plots the coverage	Checks that coverage is plotted only if there are modules	Unit test function in code	Passed

Table 6.2: Module/System Tests for Coverage Model

Function	**Function Description**	**Module/system Test Description**	**Technique Used**	**Results**
CM-45-A `plotErrorMesh()`	Plot for different mesh sizes	Checks error reduction by increasing mesh size	Module test function in code	Passed
CM-46-A `plotChangeInA()`	Plot coverage for changing semimajor axis	Performs a sensitivity analysis on a	Module test function in code	Passed
CM-47-A `plotChangeInE()`	Plot coverage for changing eccentricity	Performs a sensitivity analysis on e	Module test function in code	Passed
CM-48-A `plotChangeInI()`	Plot coverage for changing inclination	Performs a sensitivity analysis on i	Module test function in code	Passed
CM-49-A `plotChangeInHeight()`	Plot coverage for changing height	Performs a sensitivity analysis on height	Module test function in code	Passed
CM-50-A `testLessThanRadius()`	Coverage from a satellite	Checks error raises when inside Earth	Extreme value test function in code	Passed
CM-51-A `testCoverageCloseToMoon()`	Coverage from a satellite	Checks no coverage is visible when too close to Moon	Extreme value test function in code	Passed
CM-52-A `testCoverageFarFromMoon()`	Coverage from a satellite	Checks approx half of the Moon's surface is visible when far away	Extreme value test function in code	Passed
CM-53-A `plotFolds()`	Creates visual model of coverage	Checks that folds are performed correctly when there is superposition by modules	System test function in code	Passed

A set of validation codes are seen below. Specifics on the results from the tests can be seen in the main report. The test perform cross-validation between models.

Table 6.3: Validation table for the coverage model.

Function	**Validation Test Description**	**Technique Used**	**Results**
CM-54-A `PlotModelFromSOC1()`	Plot the first model from streets of coverage	Visual validation test	Passed
CM-54-B `PlotModelFromSOC2()`	Plot the second model from streets of coverage	Visual validation test	Passed
CM-54-C `PlotModelFromSOC3()`	Plot the third model from streets of coverage	Visual validation test	Passed
CM-54-D `PlotModelFromSOC5()`	Plot the fourth model from streets of coverage	Visual validation test	Passed
CM-55-A `PlotModelFromPaper1()`	Plot the first model from the optimisation paper	Visual validation test	Passed
CM-55-B `PlotModelFromPaper2()`	Plot the second model from the optimisation paper	Visual validation test	Passed

User Error Calculator

Table 6.4: Unit and Module Tests

Function	Function Description	Unit Test Description	Technique Used	Result
UE-1-A pos_dist ()	Checks if the distance from the user to the satellites is positive	Test the sign of the elements in the distance array.	Python Unittest Module	Passed
UE-2-A inv_sat_ input()	Tests initialization with less than 4 satellites	Checks if valueerror is raised	Python Unittest Module	Passed
UE-3-A inv_user_ input()	Tests initialization with less than 2 user coordinates	Checks if valueerror is raised	Python Unittest Module	Passed
UE-4-A inv_error_ input()	Tests initialization with allowable error not being size 6	Checks if valueerror is raised	Python Unittest Module	Passed
UE-5-A inv_pos_ input()	Tests initialization with negative position error	Checks if valueerror is raised	Python Unittest Module	Passed
UE-6-A satellite_ error()	Test if the UERE is a useable value	Checks if value is a float and above 0	Python Unittest Module	Passed
UE-7-A param_co variance()	Test the type for parameter covariance matrices	Checks if the param eter covariance matri ces are arrays	Python Unittest Module	Passed
UE-8-A allowable_ error()	Tests if the allowable errors are valid	Checks the sign of each allowable error which should be +	Python Unittest Module	Passed
Module Tests				
UE-9-A allowable_ error()	Tests if the allowable errors are valid	Checks the sign of each allowable error which should be +	Python Unittest Module	Passed
UE-10-A allowable_ error()	Tests if the user errors are valid	Checks the sign of each user error which should be +	Python Unittest Module	Passed
UE-11-A allowable_ error()	Test if GDOP is largest out of all DOP	Compares GDOP to each DOP and see which is greater	Python Unittest Module	Passed

Chapter 7

Navigation

Table 7.1: Unit Test for Orbits and Navigation Service Design

Function	Function Description	Unit Test Description	Technique Used	Results
NAV-01-A test_true_ anomaly_ translation()	Adding an angle to the true anomaly in order to increase it to verify coverage.	Uses the function on the Kepler elements of a satellite and compares the output with the already deviated angle.	Unit test in code	Same Kepler elements for both arrays.
NAV-01-B test_true_ anomaly_ translation_ invalid_change()	Adding an angle to the true anomaly in order to increase it to verify coverage.	Checks if the correct inputs are taken (not strings for example).	Unit test function in code.	Passed.
NAV-02-A test_model_ adder()	A function that allows for singular satellites to be added to the static model	Tests if the correct amount of satellites are added to the model	Unit test function in code.	Passed
NAV-03-A test_model_ symmetrical_ planes()	Adding multiple orbital planes with the same Kepler elements except right ascension of the ascending node. Then adding the satellites on each plane equally spaced.	Checks the number of satellites.	Unit test function in code	Passed
NAV-04-A Test_DOP_ Calculator()	Function that calculates DOP values for all 10000 points on the moon	Test analyses the size of the output to check if there are 6 values for all the points	Unit test function in code	Passed
NAV-05-A test_dyn_sim()	Function that runs the dynamic simulation so that values can be used after	Checks if the time of the function is correct by checking the shape of the states array	Unit test function in code	Passed

Table 7.2: Unit Test for Orbits and Navigation Service Design (2)

Function	**Function Description**	**Unit Test Description**	**Technique Used**	**Results**
NAV-06-A test_boxplot_ no_array()	Function that plots the ninetieth percentile of the DOP values in time in boxplots.	Checks invalid inputs for the boxplots plotting.	Unit test function in code.	Passed

Table 7.3: System Test for Orbits and Navigation Service Design

Function	**Function Description**	**System Test Description**	**Technique Used**	**Results**
NAV-07-A test_DOP_ TIME() System test	Function that uses the dynamic simulation to compute the DOPs in time.	Run the dynamic simulation, run the DOPs for all time points and check the shape of the output if it matches the time steps for all points	System test function in code	Passed

Chapter 8

Sensitivity Analysis for Trade Off

Table 8.1: Unit tests for Trade Off

Function	Function Description	Unit Test Description	Technique Used	Results
ST-01-A `test_size()`	Weight and Score inputs	Checks size of vectors are consistent	Unit test function in code	Passed
ST-02-A `test_win()`	Output of function, determining winner	Checks number of wins by first design	Unit test function in code	Passed
ST-02-B `test_win()`	Output of function, determining winner	Checks number of wins by second design	Unit test function in code	Passed
ST-02-C `test_win()`	Output of function, determining winner	Checks number of draws	Unit test function in code	Passed

Chapter 9

Propulsion

Table 9.1: Unit and System Test for satellite positioning

Function	Function Description	Unit Test Description	Technique Used	Results
SP-1-A `test_delta_a`	A calculation to check that the `delta_a` is correctly computed.	Computation of the `delta_a` for one set of inputs	Hand Calculation	Passed
SP-2-A `test_delta_V`	A calculation to check that the `delta_V` is correctly computed.	Computation of the `delta_V` for one set of inputs	Hand calculation	Passed

Chapter 10

Attitude, Determination & Control

Table 10.1: Unit and System Test for the Attitude Determination & Control Design

Function	Function Description	Unit Test Description	Technique Used	Results
AA-1-A `test_ array_ Kepler()`	A script to check that the shape of the array is as expected	Check height and width array of Cartesian coordinates of constellation satellites to known value	By unit test function in code	Passed
AA-2-A `test_ array_ relay()`	A script to check that the shape of the array is as expected	Check height and width array containing relay satellite Cartesian coordinates to known value	By unit test function in code	Passed
AA-3-A `test_ array_ A_sp()`	A script to check that the shape of the array containing sunlit surface areas is as expected	Check height and width array to known value	By unit test function in code	Passed
AA-4-A `test_ array_ x_sp()`	A script to check that the shape of the array containing x-coordinate of solar pressure is as expected	Check height and width array to known value	By unit test function in code	Passed
AA-5-A `test_ array_ y_sp()`	A script to check that the shape of the array containing y-coordinate of solar pressure is as expected	Check height and width array to known value	By unit test function in code	Passed
AA-6-A `test_ pos_ InPlane()`	A script to check that all elements are positive as required for the calculated in plane distances	Checks values are larger than 0	By unit test function in code	Passed

AA-7-A `test_ pos_ InPlane()`	A script to check that all elements are positive as required for the calculated out of plane distances	Checks values are larger than 0	By unit test function in code	Passed
AA-8-A `test_ pos_ A_sp()`	A script to check that the sunlit surface area calculated is positive	Checks value is larger than 0	By unit test function in code	Passed
AA-9-A `test_ pos_ TD()`	A script to check that the disturbance torque calculated is positive	Checks value is larger than 0	By unit test function in code	Passed
AA-10-A `test_ calc_ A_sp()`	Calculates the sunlit surface areas	Compares value to hand calculated one and checks if it's equal within a certain margin	By unit test function in code	Passed
AA-11-A `test_ calc_ x_sp()`	Calculates the x-coordinate of the centre of solar pressure	Compares value to hand calculated one and checks if it's equal within a certain margin	By unit test function in code	Passed
AA-11-A `test_ calc_ y_sp()`	Calculates the x-coordinate of the centre of solar pressure	Compares value to hand calculated one and checks if it's equal within a certain margin	By unit test function in code	Passed
AA-12-A `test_ calc_ Per()`	Calculates the orbital period of largest orbit in terms of semi-major axis	Compares value to hand calculated one and checks if it's equal within a certain margin	By unit test function in code	Passed
AA-13-A `test_ calc_ Ig()`	Calculates the MMOI	Compares value to hand calculated one and checks if it's equal within a certain margin	By unit test function in code	Passed
AA-14-A `test_ calc_ slew()`	Calculates the torque required to perform a slew manoeuvre	Compares value to hand calculated one and checks if it's equal within a certain margin	By unit test function in code	Passed
AA-15-A `test_ calc_ dump()`	Calculates the thrust needed to desaturate wheel	Compares value to hand calculated one and checks if it's equal within a certain margin	By unit test function in code	Passed

AA-15-B `test_calc_dump()`	Calculates the propellant needed to perform desaturation	Compares value to hand calculated one and checks if it's equal within a certain margin	By unit test function in code	Passed
AA-16-A `test_calc_Mom()`	Calculates the momentum storage required to counteract disturbance torque	Compares value to hand calculated one and checks if it's equal within a certain margin	By unit test function in code	Passed
AA-17-A `assign_Zero_dim()`	performs extreme value test by assigning one of the values zero. The combined with the calculation unit test, performs the zero test.	Essentially creates a plate.	By unit test function in code	Passed
System Tests				
AA-18-A `test_sensitivityrefl()`	A script that varies the reflectivity.	Sensitivity test for effect of reflectivity on the disturbance torque constants	By system test function in code	Passed
AA-19-A `test_planes`	A script that checks that no two orbital planes are the same when being fed to the pointing accuracy calculator.	checks the arrays that describe the orbital planes	By system test function in code	Passed

Chapter 11

Electrical Power System

Table 11.1: Unit Tests for Electrical Power System Design

Function	Function Description	Unit Test Description	Technique Used	Results
EP-1-A `eclipse _angle()`	Compute angle of the shadow created by an occulting body	Compare angle of hand calculation and function	By hand	Same result
EP-2-A `eclipse _length()`	Compute eclipse time due to an occulting body of a circular orbit	Compare to other function	Unit test function in code	Passed
EP-3-A `eclipse_length _ellip()`	Compute eclipse time for an elliptical orbit	Compare to circular orbit in extreme value test, by setting eccentricity to zero	Extreme value test	Passed
EP-3-B `eclipse_lenght _ellip()`	Compute eclipse time for an elliptical orbit	Compare eclipse angles of hand calculation and function	By hand	Same value
EP-4-A `sa_size()`	Calculates the area of the solar array	Check sensitivity to inputs	Sensitivity analysis in code	Behaves as expected
EP-5-A `battery_size()`	Calculates the capacity, mass, and volume of a battery	Check total battery capacity	Unit test function in code	passed
EP-5-B `battery_size()`	Calculates the capacity, mass, and volume of a battery	Check total battery mass compared to hand calculation	By hand	Same value
EP-5-C `battery_size()`	Calculates the capacity, mass, and volume of a battery	Check total battery volume compared to hand calculation	By hand	Same value
EP-6-A `charging _power()`	Calculates the power required for charging the batteries	Compare to calculation by hand	By hand	Same value

Chapter 12

Thermal Control System

Table 12.1: Unit and System Test for the Thermal Control System Design

Function	Function Description	Unit Test Description	Technique Used	Results
TC-1-A `heat_dissipated()`	A script to compute the heat dissipated by a component or subsystem.	Check dissipated heat is correct using dummy values.	Python Unittest Module	Passed
TC-2-A `visibility_factor()`	A script to compute the visibility of the spacecraft in orbit.	Check visibility is correct using dummy values.	Python Unittest Module	Passed
TC-3-A `albedo_radiation()`	A script to compute the albedo radiation.	Check albedo radiation is correct using dummy values.	Python Unittest Module	Passed
TC-4-A `intensities()`	A script to compute intensity of radiation by a radiating body.	Check intensity is correct using dummy values.	Python Unittest Module	Passed
TC-5-A `equilibrium_temperature()`	A script to compute the equilibrium temperature inside the spacecraft.	Check equilibrium temperature is correct using dummy values.	Python Unittest Module	Passed
TC-6-A `q_in()`	A script to compute the amount of heat that is necessary to keep the spacecraft on a desired temperature.	Check incoming heat is correct using dummy values.	Python Unittest Module	Passed
TC-7-A `phase_change()`	A script to compute the mass of the phase the change material.	Check if the mass is correct using dummy values.	Python Unittest Module	Passed
TC-8-A `Q_PC_day()`	A script to compute the required charging power for the phase change material.	Check if the charging power is correct using dummy values.	Python Unittest Module	Passed
System Tests				
TC-9-A `main()`	A script to compute and plot the temperature and absorbed/dissipated heat in function of the absorptivity and emissivity constants.	Sensitivity test for the absorptivity/emissivity constants and system test as it includes previously described functions.	Python Script	Passed

Chapter 13

Structures

Table 13.1: Tests for Stage 1

Function	Function Description	Unit Test Description	Technique Used	Results
ST-01-A `cyl_volume()`	Computes the volume of a body	Checks correct volume	Unit test function in code	Same value
ST-02-A `cyl_surface()`	Computes the surface of a body	Checks correct surface area	By hand	Same value
ST-03-A `cyl_cross()`	Computes the cross-sectional area of the body		Unit test function in code	Same value
ST-04-A `cyl_areaI()`	Computes the area moment of inertia	Checks the correct I	Unit test function in code	Same value
ST-05-A `cyl_MMOI()`	Computes the MMOI of the shape	Checks if the MMOI is correctly calculated	Unit test function in code	Same value
ST-06-A `cyl_inputs()`	Initialises the shape object	Gives incorrect inputs to the initializer	Unit Test	An exception is raised
ST-07-A `prism_volume()`	Computes the volume of a body	Checks correct volume	Unit test function in code	Same value
ST-8-A `prism_surface()`	Computes the surface of a body	Checks correct surface area	Unit test function in code	Same value
ST-09-A `prism_cross()`	Computes the cross-sectional area of the body		Unit test function in code	Same value
ST-10-A `prism_areaI()`	Computes the area moment of inertia	Checks the correct I	Unit test function in code	Same value
ST-11-A `prism_MMOI()`	Computes the MMOI of the shape	Checks if the MMOI is correctly calculated	Unit test function in code	Same value
ST-12-A `prism_inputs()`	Initialises the shape object	Gives incorrect inputs to the initializer	Unit Test	An exception is raised
ST-13-A `s1_a_freq()`	Computes the axial frequency of the structure	Checks the frequency calculation is correct	Unit test function in code	Same value
ST-14-A `s1_l_freq()`	Computes the lateral frequency of the structure	Checks the frequency calculation is correct	Unit test function in code	Same value

ST-15-A `s1_a_stress()`	Computes the max axial stress of the structure	Checks the axial stress is correct	Unit test function in code	Same value
ST-16-A `s1_t_buck()`	Computes thickness required for buckling	Checks the thickness calculation is correct	Unit test function in code	Same value
ST-17-A `s1_char()`	Computes axial stress based on a thickness	Checks the axial stress calculation is correct	Unit test function in code	Same value
ST-17-B `s1_char()`	Computes lateral stress in x based on a thickness	Checks the lateral stress calculation is correct	Unit test function in code	Same value
ST-17-C `s1_char()`	Computes lateral stress in y based on a thickness	Checks that the lateral stress calculation is correct	Unit test function in code	Same value
ST-17-D `s1_char()`	Computes the tensile stress based on a thickness	Checks that the tensile stress calculation is correct	Unit test function in code	Same value
ST-17-E `s1_char()`	Computes the compressive stress based on a thickness	Checks that the compressive stress calculation is correct	Unit test function in code	Same value
ST-18-A `s1_panels()`	Adds a panel to the structure	Checks the moments of inertia adapt based on the shape	Unit test function in code	Same value

Table 13.2: Tests for Detailed Stage

Function	**Function Description**	**Unit Test Description**	**Technique Used**	**Results**
ST-19-A `prop_tank_volume()`	Computes the volume of the propellant tank	Checks whether the volume is calculated correctly	Unit test function in code	Same value
ST-20-A `sphere_MMOI()`	Computes the MMOI of the spherical propellant tank	Checks whether the MMOI is correctly calculated	Unit test function in code	Same value
ST-21-A `cylinder_MMOI()`	Computes the MMOI of the cylindrical propellant tank	Checks whether the MMOI is correctly calculated	Unit test function in code	Same value
ST-22-A `sphere_thickness()`	Computes the thickness needed for the spherical propellant tank	Checks correct thickness	Unit test function in code	Same value
ST-23-A `cylinder_thickness()`	Computes the thickness needed for the cylindrical propellant tank	Checks correct thickness	Unit test function in code	Same value
ST-24-A `sphere_mass()`	Computes the mass of the spherical propellant tanks	Checks the correct mass	Unit test function in code	Same value

ST-25-A `cylinder_mass()`	Computes the mass of the cylindrical propellant tanks	Checks the correct mass	Unit test function in code	Same value
ST-26-A `init_struc()`	Initialises the structure and all its properties	Checks whether all properties are initialised.	Unit test function in code	Instance created
ST-27-A `add_mass()`	Adds mass to an element	Checks whether the mass is added	Unit test function in code	Same value
ST-28-A `add_struc()`	Adds the structure to the system	Checks whether properties of the system are correctly changed	Unit test function in code	Same value
ST-29-A `add_other()`	Adds a point mass element to the system	Checks that the element is added, and the properties updated	Unit test function in code	Same value
ST-30-A `panelsMMOI()`	Adds solar panels to the structure	Checks that the solar panels modify the systems parameters	Unit test function in code	Same value
ST-31-A `add_dict()`	Uses a dictionary to create elements in the body	Checks that the elements are created correctly	Unit test function in code	Instance created
ST-32-A `change_MMOI()`	Computes the MMOI of a system with elements	Checks MMOI updates after adding a value	Unit test function in code	Same value
ST-33-A `test_stress()`	Computes the maximum stresses in the system	Checks the values are correct	Unit test function in code	Same value
ST-34-A `test_vib()`	Computes the vibrations in three directions for the system	Checks values are correct	Unit test function in code	Same value
ST-35-A `test_sys()`	Ensures compliance with limiting factor (yield)	Checks if boolean is consistent	System test function in code	Same boolean
ST-35-B `test_sys()`	Ensures compliance with limiting factor (axial vibration)	Checks if boolean is consistent	System test function in code	Same boolean
ST-35-C `test_sys()`	Ensures compliance with limiting factor (lateral vibration)	Checks if boolean is consistent	System test function in code	Same boolean
ST-35-D `test_sys()`	Ensures compliance with limiting factor (buckling)	Checks if boolean is consistent	System test function in code	Same boolean
ST-35-E `test_sys()`	Ensures compliance with limiting factor (shear stress)	Checks if boolean is consistent	System test function in code	Same boolean
ST-36-A `sens_t1()`	Calculates the thickness of a spherical propellant tank	Plots sensitivity analysis of thickness	Sensitivity Analysis Test in Code	Consistent Figure

ST-37-A `sens_t2()`	Calculates the thickness/length of a cylindrical propellant tank	Plots length vs mass	Sensitivity Analysis Test in Code	Consistent Figure
ST-38-A `sens_dMMOI()`	Adds a point mass to the system	Plots distance from origin vs MMOI	Sensitivity Analysis Test in Code	Consistent Figure
ST-39-A `sens_solarMMOI()`	Adds a solar panel to the systems	Plots MMOI changes vs solar panel area	Sensitivity Analysis Test in Code	Consistent Figure